مقدمة إلى:

برمجة التطبيقات الشبكية باستخدام تقنية JAVA RMI

م. محمد العليان

@mhdalyan

الترخيص

للتواصل مع الكاتب

LinkedIn

about.me

Twitter

مقدمة

يُلخص هذا الكتيب التعليمي مفاهيم أساسية في برمجة التطبيقات الموزعة باستخدام تقنية Java RMI، وهو موجه لطلاب كليات علوم الحاسب، والهندسة المعلوماتية، والمبرمجين المهتمين بهذه التقنية والذين يتقنون لغة البرمجة جافا.

يُقدم هذا الكُتيب التعليمي البسيط والمتماسك نظرة عامة على إحدى تقنيات البرمجة المستخدمة في النظم الموزعة و هي Java RMI، يحتوي الكُتيب على شرح لبعض المفاهيم النظرية المستخدمة في هذا النموذج من البرمجة الموزعة والمسمى Remote Reference Module، وذلك من خلال مثال عملي بسيط يشرح الفكرة ببساطة، بالإضافة إلى كيفية تنفيذ المثال و مشاهدة الخرج وذلك من خلال موجه الأوامر Command Line.
ثم الإنتقال لدراسة حالة أعمق وأكبر وهي تطبيق محادثة شبيه ببرنامج سكايب الشهير وقد فَصلّت في شرح الركائز الأساسية في بناء هذا التطبيق،كما يمكنكم مشاهدة كود التطبيق على منصة GitHub ، والتي تتيح مشاركة الكود وإعادة استعماله بشكل اجتماعي وتعاوني.
سأكون مسروراً حقاً بملاحظاتكم على هذا الكتيب، وأرجو ألا تبخلو بها.
آمل من الله تعالى أن يكون هذا الكُتيب مفيداً لكم وأن يقدم العون إلى كل من يريد أن يتعلم هذه التقنية، وأرجو أن يكون عملي هذا في صحيفة أعمالي، والله من وراء القصد.

دمشق في 29-5-2014

محمد العليان

جدول المحتويات

مقدمة تاريخية

مع تطور الحياة اليومية و ظهور الحوسبة الموزعة (Distributed Computing)، ظهرت الحاجة إلى التخاطب بين التطبيقات و خصوصاً التخاطب بين التطبيقات القديمة (Legacy Systems) والحديثة.
عملية التخاطب هذه تتم باستخدام برمجيات وسيطة أو ما تسمى Middleware، وهذه البرمجيات الوسيطة هي عبارة عن معيار تخاطب بين التطبيقات له خصائص معينة، وهذه الخصائص في الحقيقة هي متطلبات غير وظيفية ، أي أنها مجموعة من الشروط والقيود مثل: أن يعمل التطبيق على الويب (Web Based) و الحماية (Security) وأن تكون مستقلة عن نوع الحاسب وعن نظام التشغيل (Platform independent). والكثير من المزايا الأخرى وجميعها تشترك في أنها خصائص غير وظيفية.
سنتكلم عن Middleware خاصة بشركة Sun Microsystems[1] ، هي RMI (**R**emote **M**ethod **I**nvocation) وهي خاصة بلغة جافا حصراً، وتتميز هذه التكنولوجيا بسهولتها وقوتها وهذا ما سنراه لاحقاً.

مقدمة عن RMI

RMI هي تقنية خاصة بلغة جافا وهي مستخدمة في النظم الموزعة، هذه التقنية تسمح لغرض Object يعمل على JVM على حاسب ما أن يستدعي (Invoke) لطريقة (Method) غرض أخر يعمل على JVMعلى حاسب آخر، وهذان الحاسبان موصولان بشبكة طبعاً.

- هذه التقنية مفيدة جداً في تطوير الأنظمة الكبيرة التي تحتاج إلى قابلية التوسع Scalability[2].
- تسمح لنا RMI باستدعاء مناهج Methods لأغراض موجودة على حاسب بعيد كما لو أنها موجودة على الحاسب المحلي.

كل RMI Service يتم تعريف واجهة لها Interface، والتي توصف جميع مناهج الأغراض التي يمكن ان تستدعى عن بعد.هذه الواجهة يجب أن تكون مشتركة بالنسبة لكل المطورين الذين سيكتبون هذه الخدمة.أي إن هذه الواجهة تتصرف كما لو أنها مخطط للتطبيقات التي سوف تستخدم وتُزود التوليفات implementation لهذه الخدمة عن طريق المطورين.

مقارنة بسيطة بين RMI و RPC

في الحقيقة RPC ظهرت بعد أن انتشرت تقنيات البرمجة غرضية التوجه OOP، وبما أن RPC تستخدم في نمط البرمجة الأجرائية قدمت بعض الشركات مثل مايكروسوفت بروتوكول سمته DCOM ، وقامت Sun Microsystems بتقديم JAVA RMI، بينما قدمت مجموعة OMG بروتوكول CORBA وبذلك أصبح من الممكن استخدام RPC في لغة غرضية التوجه.
مشكلة RMI أنها تدعم البرامج المكتوبة بلغة جافا فقط.

ملاحظة: في الحقيقة مع ظهور J2EE قدمت شركة Sunحلاً لهذه المشكلة من خلال تكنولوجيا تسمى RMI over IIOP[3]، والتي هي عبارة عن جسر ربط مع أنظمة CORBA.

---

[1] حالياً Oracle لأن Oracle اشترت شركة Sun.

[2] التأقلم مع إزدياد الضغط عن طريق إضافة حاسب جديد.

[3] IIOP هو بروتوكول CORBA.

كيف تعمل RMI

- الأنظمة التي تستخدم RMI كوسيلة للتواصل بين التطبيقات تُقسم عادة إلى مجموعتين : الزبائن Clients والمخدمات Servers.المخدم Server يقدم خدمة RMI والزبون يقوم باستدعاء منهج غرض من هذه الخدمة.

- مخدمات RMI يجب أن تسجل نفسها ضمن خدمة rmiregistry تؤمن جدول تقابل(Lockup) لهذه الخدمات، وذلك لكي تسمح للزبائن بأن يجدوا هذه الخدمات، أو يمكنهم أن يحصلوا على مرجع Reference متوفر لهذه الخدمة بطريقة ما.

- يأتي مع الجافا تطبيق يُسمى rmiregistry، والذي يعمل كمهمة Process مستقلة ويسمح للتطبيقات بأن تسجل خدمات RMI أو أن تحصل على مرجع Reference لأسم خدمة معينة.
  حالما يقوم المخدم بالتسجيل سوف يقوم بإنتظار طلبات الزبائن من أجل خدمة RMI معينة.
  كل عملية تسجيل خدمة في rmiregistry مرتبطة باسم هذه الخدمة والتي يُعبر عنها بـ String لها عنوان URL (uniform Resource Locater) توصف هذه الخدمة ولكي يسمح للزبائن بأن يختاروا الخدمة المناسبة عن طريق جدول التقابل الموجود في rmiregistry .

- الزبون يرسل رسالة RMI ليستدعي منهج غرض عن بعد. قبل أن يحدث هذا يجب على الزبون أن يحصل على Reference للكائن البعيد، و يتم الحصول عليه من خلال خدمة جدول التقابل Lockup Service الموجودة داخل rmiregistry.

- التطبيق الزبون يطلب اسم خدمة معينة وينشئ URL تُمثل مرجع إلى الكائن البعيد Remote Object Reference.

  وطبعاً يجب أن نعلم أن URL ليس خاصاً بالبروتوكول HTTP وإنما يمكن ذلك مع أي ملف على الحاسب (له URL معين).طبعاً RMI تستخدم String تخضع لقواعد URL لتمثيل مرجع إلى الكائن البعيد Remote Object Reference ولها الشكل التالي :

  Rmi://hostname:port/servicename

  حيث :

  1- Hostname: اسم المخدم أو الـــ IP الخاص به.
  2- port : عنوان التطبيق الذي يؤمن هذه الخدمة على حاسب المخدم(إختياري).
  3- Servicename: اسم الخدمة وهو عبارة عن اسم الخدمة أو واجهة الخدمة Service Interface كما سنرى في الكود لاحقاً.

- حالما يتم الحصول على المرجع Reference الذي يشير إلى الكائن البعيد وذلك عن طريق rmiregistry، عندها يستطيع الزبون أن يتفاعل مع الخدمة البعيدة (يقوم باستدعاء المناهج المعرُفة داخل واجهة الخدمة عن بعد وكأنه يستدعيها على الحاسب الخاص به).

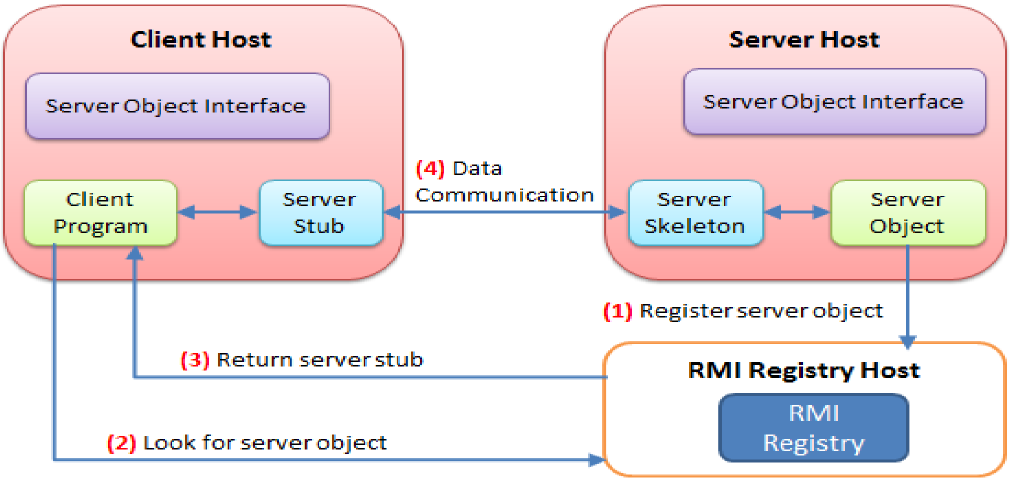
- التفاصيل الشبكية المتعلقة بكيفية إرسال الطلب من الزبون هي شفافة تماماً بالنسبة لمطور التطبيق.

- لقد أصبح العمل مع الكائنات البعيدة بسيط جداً كما لو أننا نعمل مع كائنات محلية، وذلك بسبب التقسيم الذكي المستخدم في نظام RMI وهذا التقسيم مؤلف من مكونين هما :

1- Stub : عبارة عن غرض يقوم بإيصال الطلبات إلى المخدم البعيد. يقوم Stub بفتح Socket مع المخدم البعيد يحوي طلب الخدمة[1] ، ثم يقوم بالإنتظار حتى يتم استدعاء المنهج ثم يرد النتائج إلى الـــ Stub والذي يعيدها إلى ال Method المستدعي.

2- Skeleton: عبارة عن غرض يمثل Listener يقوم بالتنصت على Port معينة ليستقبل طلبات الزبائن(والتي يتم إرسالها عن طريق الـــStub) ومن ثم يقوم بتمرير هذه الطلبات إلى RMI Services.

عملية التواصل بين الـــ Stub عند الزبون وال Skeleton عند المخدم تتم باستخدام TCP Socket.
يمكن تلخيص آلية عمل stub و skeleton من خلال الصورة التالية :



تعريف واجهة خدمة RMI

أي نظام سوف يستخدم RMI يجب أن يستخدم واجهة الخدمة Service Interface، وهي عبارة عن تعريف لمناهج الغرض التي يمكن استدعائها عن بعد. بالإضافة إلى توصيف البارمترات والقيمة المُعادة من المناهج وحتى الاستثناءات التي يجب أن يعالجها المنهج. تحقق RMI Service ايضاً Implementation لهذه الواجهة عن طريق صف (Class) ما. لهذا السبب يقوم المطورون بإعادة تعريف المناهج Methods سلفاً، وايقاف التغيرات على الواجهة (Interface) حتى تبدأ عملية التطوير.

كل واجهات الخدمة Service Interfaces يجب أن ترث من الواجهة Java.rmi.Remote والتي تساعد في تعريف المناهج التي يمكن أن تُستدعى عن بعد.

ملاحظة : كلمة Service Interface تكافىء Service name لأن الخدمة يتم تعريفها كواجهة.

مثال :
سنقوم بتعريف الواجهة RMILightBulb والتي تمثل فعلياً خدمة RMI والتي تعبر عن حالة مصباح (يعمل أو لا يعمل). في الحقيقة إن أي خدمة يقدمها المخدم هي عبارة عن واجهة ترث من الواجهة Java.rmi.Remote .

[1] لأن RMI مبنية فوق RPC و RPC مبنية فوق Socket.

```java
//Service interface
public interface RMILightBulb extends java.rmi.Remote
{
	public void on ()   throws java.rmi.RemoteException;
	public void off()   throws java.rmi.RemoteException;
	public boolean isOn() throws java.rmi.RemoteException;
}
```

هذه الواجهة تحوي المناهج التي يجب أن تكون متوفرة في أي مصباح أو في أي شي يضيئ.

تحقيق واجهة خدمة RMI

حالما يتم تعريف واجهة الخدمة يجب أن نقوم بتحقيق هذه الخدمة (واجهة الخدمة)عن طريق إعادة كتابة جميع الوظائف (Overriding) الموجودة في هذه الواجهة من خلال Class معين، لنرى الكود التالي :

```java
public class  RMILightBulbImpl  extends java.rmi.server.UnicastRemoteObject
	implements RMILightBulb
{
	// A constructor must be provided for the remote object
	public RMILightBulbImpl() throws java.rmi.RemoteException
	{
		// Default value of off
		setBulb(false);
	}
	// Boolean flag to maintain light bulb state information
	private boolean lightOn;
	// Remotely accessible "on" method - turns on the light
	public void on() throws java.rmi.RemoteException
	{
		// Turn bulb on
		setBulb (true);
	}
	// Remotely accessible "off" method - turns off the light
	public void off() throws java.rmi.RemoteException
	{
		// Turn bulb off
		setBulb (false);
	}
	// Remotely accessible "isOn" method, returns state of bulb
	public boolean isOn() throws java.rmi.RemoteException
	{
		return getBulb();
	}
	// Locally accessible "setBulb" method, changes state of bulb
	public void setBulb (boolean value)
	{
		lightOn = value;
	}
	// Locally accessible "getBulb" method, returns state of bulb
	public boolean getBulb ()
	{
		return lightOn;
	}
}
```

إنشاء الـ Stub and Skeleton Classes

إنشاء الصفوف الخاصة **Stub** والـ **Skeleton** هي ليست من مهام المطور وإنما عن طريق مترجم RMI، وهذا المترجم يُسمى **rmic** والتي تأتي كجزء من حزمة الـــ JDK. ويتم ذلك عن طريق موجه الأوامر Command line كما يلي :

```
rmic  implementation
```

حيث implementation هي اسم الصف الذي يحقق الخدمة(واجهة الخدمة RMILightBulb) وهي في حالتنا الصف RMILightBulbImpl

وبالتالي نكتب في موجه الأوامر :

```
rmic  RMILightBulbImpl
```

يجب أن يكون كل من واجهة الخدمة RMILightBulb والصف الذي يحققها RMILightBulbImpl قد تمت ترجمتهما (نتج لدينا ملفات بلاحقة .class) حتى يتم توليد الملفين التالين :

```
RMILightBulbImpl_Skel.class
RMILightBulbImpl_Stub.class
```

إنشاء تطبيق مخدم RMI

المخدم هو المسؤول عن إنشاء كائن من الصف الذي يحقق الخدمة[1]، ومن ثم يقوم بعملية تسجيل الخدمة ضمن rmiregistry.

لنرى الكود التالي :

```
import java.rmi.*;
import java.rmi.server.*;
public class LightBulbServer
{
        public static void main(String args[])
        {
                System.out.println ("Loading RMI service");
                try
                {
                         // Load the service
                        RMILightBulbImpl bulbService = new RMILightBulbImpl();
                        // Examine the service, to see where it is stored
                        RemoteRef location = bulbService.getRef();
                        System.out.println (location.remoteToString());
                        // Check to see if a registry was specified
                        String registry = "localhost";
                        String registration = "rmi://" + registry + "/RMILightBulb";
                        // Register with service so that clients can find us
                        Naming.rebind(registration, bulbService );
                }
                catch (RemoteException re)
                {
                         System.err.println ("Remote Error - " + re);
                }
                catch (Exception e)
                {
```

[1] عندما نتكلم عن خدمة فأننا نقصد واجهة الخدمة Service Interface

```
                              System.err.println ("Error - " + e);
                        }
                  }
      }
```

كيفية العمل

في البداية قمنا بإنشاء كائن من الصف الذي يحقق الخدمة والذي هو الصف RMILightBulbImpl، ثم قمنا بالحصول على مرجع إلى الكائن البعيد Reference to Remote Object عن طريق المنهج getRef() والذي يظهر فقط عند تحقيق الواجهةRemote، ثم قمنا بتشكيل URL تُمثل مرجع إلى الكائن البعيد.

الآن سنقوم بعملية تسجيل الخدمة، يتم تسجيل الخدمة إعتماداً على الصف Naming وهو صف مسؤول عن تسجيل خدمة من قبل مخدم أو طلب خدمة من قبل زبون. يتم تسجيل الخدمة المنهج rebind و هو منهج static يأخذ بارمترين الأول هو registration وهو متحول String يوصف الخدمة بشكل كامل ويتم إنشاؤه عند تسجيل خدمة من قبل المخدم أو طلب خدمة معينة من قبل الزبون، يخضع هذا المتحول إلى قواعد URL فيحوي عنوان الحاسب الذي يستضيف الخدمة بالإضافة إلى اسم الخدمة(وهي فعلياً اسم واجهة الخدمة Service Interface)، أما الـــ Port فهو اختياري. هذا المتحول registration يمثل اسم خدمة سيتم إضافتها إلى rmiregistry(وقد تكون موجودة). هذا الاسم يخضع لصيغة URL ويمثل مرجع لكائن بعيد.
أما البارمتر الثاني :فهو يُمثل الكائن البعيد الذي سيشير إليه البارمتر الأول(المتحول registration).
وطبعاً لا ننسى موضوع معالجة الإستثناءات فهو هام جداً لإستقرار البرنامج ولمعرفة مكان وجود المشكلة في حال حدوثها أثناء التنفيذ.

إنشاء تطبيق زبون RMI

في الحقيقة كتابة تطبيق زبون RMI أبسط من كتابة تطبيق مخدم. كل ما يحتاجه الزبون هو الحصول على مرجع إلى الكائن البعيد، وهذا يتم من خلال إنشاء متحول String يوصف الخدمة بشكل كامل و هذا التوصيف يخضع إلى قواعد URL (كما في المخدم)، ثم نقوم بالبحث عنه ضمن rmiregisrty الخاص بالمخدم وذلك عن طريق إرسال طلب إلى المخدم (بواسطة الـــ Stub) والذي يستقبل هذا الطلب هو الـــ Skeleton والذي يعمل كـ Listener يستقبل الطلبات ثم يمررها لخدمة RMI.
ثم نحصل على مرجع إلى الكائن البعيد و نستدعي ما نشاء من المناهج المُعرفة ضمن واجهة الخدمة (جميع الإستدعاءات تتم عن بعد).لنرى الكود التالي :

```
public class LightBulbClient
{
      public static void main(String args[])
      {
            System.out.println ("Looking for light bulb service");
            try
            {
                  // Check to see if a registry was specified
                  String registry = "localhost";
                  // Registration format //registry_hostname (optional):port /service
                  String registration = "rmi://" + registry + "/RMILightBulb";
                  // Lookup the service in the registry, and obtain a remote service
                  Remote remoteService = Naming.lookup ( registration );
                  // Cast to a RMILightBulb interface
                  RMILightBulb bulbService = (RMILightBulb) remoteService;
                  // Turn it on
                  System.out.println ("Invoking bulbservice.on()");
                  bulbService.on();
                  // See if bulb has changed
                  System.out.println ("Bulb state : " + bulbService.isOn()  );
                  // Conserve power
                  System.out.println ("Invoking bulbservice.off()");
                  bulbService.off();
```

```java
                // See if bulb has changed
                System.out.println ("Bulb state : " + bulbService.isOn() );
            }
            catch (NotBoundException nbe)
            {
                System.out.println ("No light bulb service available  in registry!");
            }
            catch (RemoteException re)
            {
                System.out.println ("RMI Error - " + re);
            }
            catch (Exception e)
            {
                System.out.println ("Error - " + e);
            }
        }
    }
```

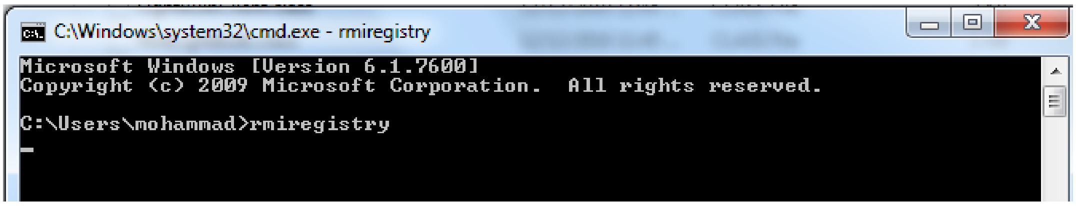
كيفية العمل

كل ما سيقوم به الزبون الآن هو تشكيل الـــ registration التي تخضع لقواعد URL والبحث عنها ضمن الـــ rmiregistry عن طريق المنهج Naming.lockup(registration)، وبالتالي نحصل على مرجع إلى الكائن البعيد[1] ، ونضعه ضمن مرجع Reference من الواجهة Remote ، ثم نقوم بعلمية Cast على الواجهة RMILightBulb. و نقوم باستدعاء مجموعة من المناهج الخاصة بالكائن البعيد (والذي يحقق الواجهة RMILightBulb).

نلاحظ أن الزبون لا يحتاج إلى نسخة من الـــ Implementation الخاصة بالكائن البعيد وإنما يكفيه فقط مرجع Reference من الواجهة التي يحققها الكائن البعيد(RMILightBulb)، والتي هي في الحقيقة تمثل الخدمة بحد ذاتها، وعن طريق الـــ Upcasting يتم استدعاء المناهج عن بعد.

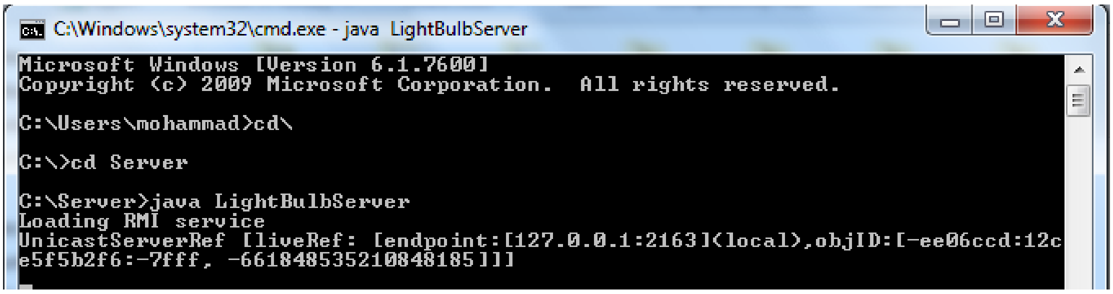
خطوات تشغيل النظام

نشغل الخدمة rmiregistry (و هي عبارة عن Process مستقلة) من موجه الأوامر كما يلي :

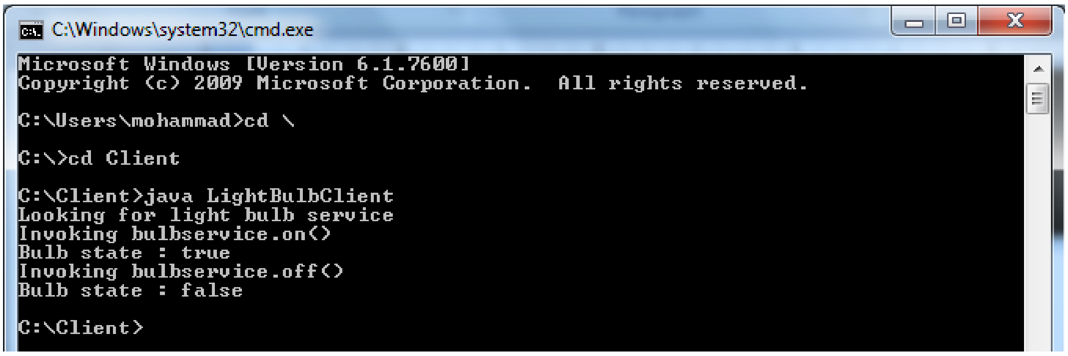


نشغل برنامج المخدم(ليقوم بتسجيل الخدمة ضمن **rmiregistry**)

[1] الغرض bulbService الذي قام المخدم قبل قليل بتمريره إلى المنهج الساتيكي rebind() عندما قام المخدم بعملية تسجيل الخدمة.



نشغل برنامج الزبون (ليطلب الخدمة من **rmiregistry**)



كل ما قمنا به من تشغيل لــ rmiregistry و توليد stub, skeleton سيتم توليده تلقائياً عند استعمال بيئة التطوير المتكاملة Netbeans IDE حيث أنها تولد stub, skeleton وأما تشغيل rmiregistry فإننا سنقوم بتشغيليه عن طريق كود سنراه لاحقا.

حالة دراسية، تطبيق شبيه بنظام Skype

سنقوم ببناء نظام محادثة موزع يعتمد على فكرة التخاطب المباشر بين الأشخاص المتواجدين في نفس غرفة المحادثة وذلك بالإعتماد على تقنية Java RMI.

سيكون لدينا بشكل رئيسي تطبيقين هما المخدم والزبون، تطبيق المخدم يقوم بتعريف واجهة الخدمة تقوم بعدة وظائف منها إضافة أو حذف غرفة محادثة، تسجيل عضوية مستخدم، تسجيل دخول وخروج على مجموعة محددة، وغيرها من الوظائف الموجودة في أنظمة المحادثة المتوفرة.

يقوم تطبيق الزبون باستعمال كافة الوظائف الموجودة في واجهة الخدمة الموجودة لدى المخدم، بالإضافة إلى أنه يقوم بتعريف واجهة خاصة به تحوي تابع خاص باستقبال رسالة، يقوم المخدم باستدعائه عندما يقوم احد الأشخاص بارسال رسالة إلى أخر.

سنقوم لاحقاً بتطبيق مبدأ الند للند في الإرسال (peer to peer) بين الزبائن دون أن يقوم المخدم بعملية تحويل الرسائل بين الأشخاص.

RMI Chat server

يقدم المخدم العديد من الخدمات منها إنشاء وحذف غرفة محادثة، تسجيل حساب، تسجيل دخول على غرفة معينة، تسجيل خروج من غرفة معينة، ارسال رسالة إلى شخص أخر ضمن الغرفة، كما يمكن بث الرسالة إلى جميع أعضاء الغرفة.

واجهة *الخدمة*

لدينا الواجهتين التاليتين حيث يوجد نسخة منهما عند المخدم والزبون، هذه الواجهة يقوم المخدم بتنجيزها(implementation) عن طريق الصف Server_Services_Imp. الواجهة الأولى كما يلي :

```
public interface IServerServices extends Remote
{
    int   AddChatRoom(String name)throws RemoteException ;
    int   AddChatRoom(String name,int capacity)throws RemoteException;
    int   removeChatRoom(String name)throws RemoteException ;
    int   signup(String userName, String password ,String FName ,String LName)throws RemoteException;
    int   signin(String userName, String password,String roomName)throws RemoteException;
    int   signout(String userName,String roomName)throws RemoteException;
    String [] ListAvailableRooms()throws RemoteException;
    String [] ListClientsInRoom(String name)throws RemoteException;
    int   SendMessage(String m ,String to,String roomName)throws RemoteException;
    int   SendBroadCatMessage(String userName,String m,String roomName)throws RemoteException;
    void  AddReceiveListener(IClientListener c)throws RemoteException;
    void  RemoveReceiveListener(IClientListener c)throws RemoteException;
}
```

الواجهة الثانية يتم تنجيزها من قبل الزبون وهي تحوي المناهج التي يجب على المخدم أن يستدعيها من عند الزبون ومنها تابع الاستقبال.

```
public interface IClientListener  extends Remote
{
     void ReceiveMessage(String s)throws RemoteException;
     String GetID()throws RemoteException;
     void getListeners(Vector<IClientListener>listeners)throws RemoteException;
}
```

يقوم الزبون بإنشاء صف ClientListener يقوم بتنجيز الواجهة IClientListener، وذلك من أجل استدعاء المنهج AddReceiveListener الذي يسجل الغرض الخاص بالزبون المتصل (غرض من الصنف ClientListener)، لكي يقوم المخدم لاحقاً بإيصال الرسائل الواردة إليه عن طريق "الاستدعاء الخلفي" Call Back للمنهج ReceiveMessage الموجود ضمن الصنف ClientListener. يقوم المخدم بهذه المهمة عن طريق مرجع من الواجهة IClientListener (لديه نسخة منها).

آلية عمل التطبيق

في البداية نقوم بتشغيل المخدم الذي يقوم بإنشاء غرض من الصف Server_Services_Imp، والذي يحقق الواجهة IServerServices، ثم نقوم بتشغيل RMI Registry عن طريق التعليمة التالية والذي يتنصت على المنفذ 6000.

```
Registry reg = LocateRegistry.createRegistry(6000);
```

ثم نقوم بتسجيل الخدمة وذلك كما يلي :

```
Server_Services_Imp obj = new Server_Services_Imp();
String name = "rmiServer";
reg.rebind(name, obj);
```

يعمل المخدم الآن.

تطبيق الزبون

يقوم الزبون في البداية بالوصول إلى RMI Registry، و من ثم البحث ضمنه عن خدمة معينة، ومن ثم الوصول إلى الغرض البعيد الموجودة في ذاكرة المخدم.

```
Registry reg=LocateRegistry.getRegistry("localhost", 6000);
Remote ref=  reg.lookup("rmiServer");
remote_ref=(IServerServices)ref;
```

الآن يمكننا استدعاء المناهج الموجودة عند المخدم كما لو أنها كانت موجودة محلياً.

يقوم الزبون بعمل حساب خاص به، ومن ثم يقوم بتسجيل دخول على غرفة معينة، يقوم بعد ذلك بإنشاء غرض من الصنف ClientListener والذي يحقق الواجهة IClientListener، يحوي هذا الغرض بعض المعلومات عن الزبون الحالي، ثم يقوم الزبون بتسجيل الغرض الخاص به والذي يمثله ضمن المخدم عن طريق المنهج AddReceiveListener والذي لديه قائمة بالأغراض الخاصة بالزبائن المتصلين.

```
int flag= remote_ref.signin(Username1.getText(),password.getText(),s);
if(flag==1)
{
    ClientListener client=new ClientListener(username);
    remote_ref.AddReceiveListener(client);
}
else
    System.out.println("Error in Sign in");
```

الآن عندما يقوم أي من الزبائن المتصلين بالمخدم بإرسال رسالة، وذلك عن طريق استدعاء المنهج SendMessage الموجود في الغرض البعيد، فإن المخدم يقوم بالبحث عن الزبون الوجهة في قائمة الأغراض المسجلين (قائمة أغراض من الصنف ClientListener)، وعندما يجده يقوم باستدعاء المنهج ReceiveMessage الخاص بالغرض الوجهة ويمرر له الرسالة المرسلة من الزبون المرسل، وهذا ما نسميه بالـ CallBack Method.

نلاحظ أن أي رسالة مرسلة يجب أن تمر في البداية على المخدم، والذي هو عقدة مركزية ومن ثم يقوم المخدم بالبحث عن الوجهة ومن ثم القيام بالاستدعاء الخلفي callback للمنهج ReceiveMessage.
نريد أن نجعل الإعتماد على المخدم المركزي أقل خصوصاً فيما يتعلق بإرسال الرسائل بين الزبائن، حيث يقوم الزبون بإرسال الرسالة بشكل مباشر للطرف الأخر(الزبون الوجهة) ، بدلاً من أن يقوم بإرسالها للمخدم ومن ثم يقوم المخدم بإرسالها إلى الزبون الوجهة.

التحقيق البرمجي لهذه الفكرة يتم كما يلي

في المخدم بعد أن يقوم الزبون بتسجيل الدخول إلى الغرفة، يتم إرسال قائمة الأغراض الموجودة لدى المخدم إلى جميع الزبائن المتصلين، وذلك عن طريق Call Back للمنهج getListeners، الموجود في الغرض الخاص بالزبون الذي يريد القيام بعملية تسجيل الدخول.
عندما يريد الزبون 1 ارسال رسالة إلى الزبون 2، يقوم بالبحث عنه ضمن القائمة التي لديه، ومن ثم يتم استدعاء المنهج ReceiveMessage الخاص بالزبون الثاني من دون المرور على المخدم، وهكذا يصبح التطبيق في جزء من أجزائة Peer 2 peer.

```
String Des= "Ahmad";
clientListener = getClientListener(Des);
clientListener.ReceiveMessage(message);
```

يمكن الاطلاع على الكود البرمجي للتطبيق من خلال الرابط التالي :

https://github.com/MhdAlyan/RMIChatServer